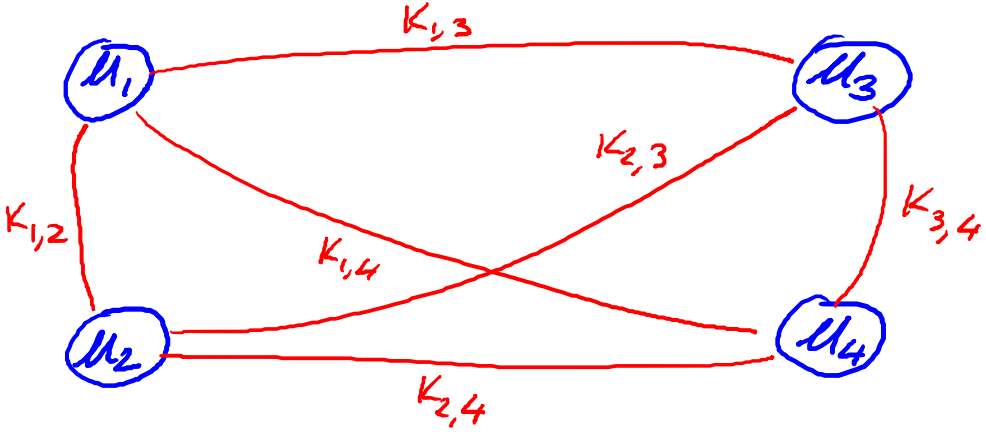
Public Key Encryption

Jong Hwan Park

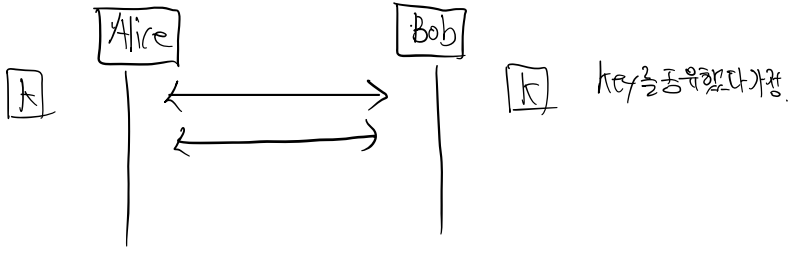
Key management

Problem: n users. Storing mutual secret keys is difficult

상호간에 키를 공유해야한다.



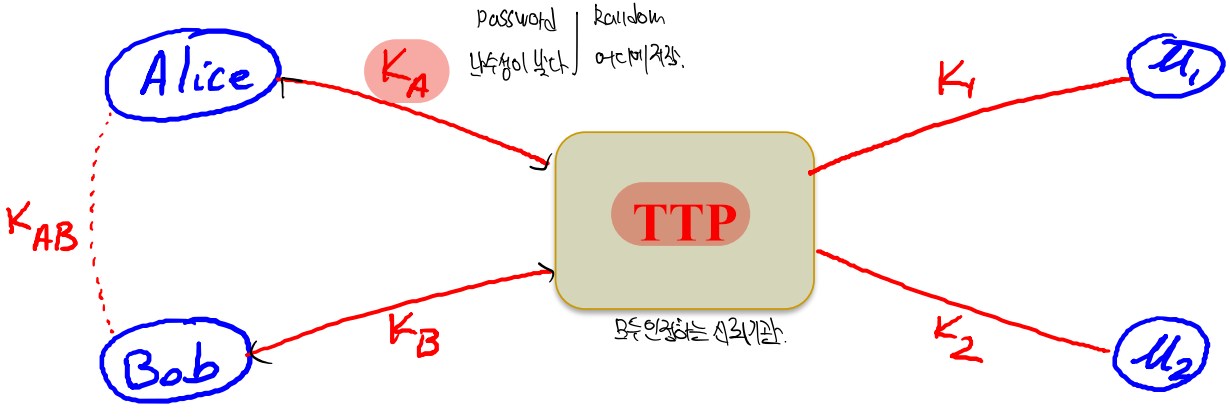
Total: O(n) keys per user



A better solution

Online Trusted 3rd Party (TTP)

앞의 단점을 극복.



password 난수성이 낮다 | Random 어디에저장.

모든인증하는 신뢰기관.

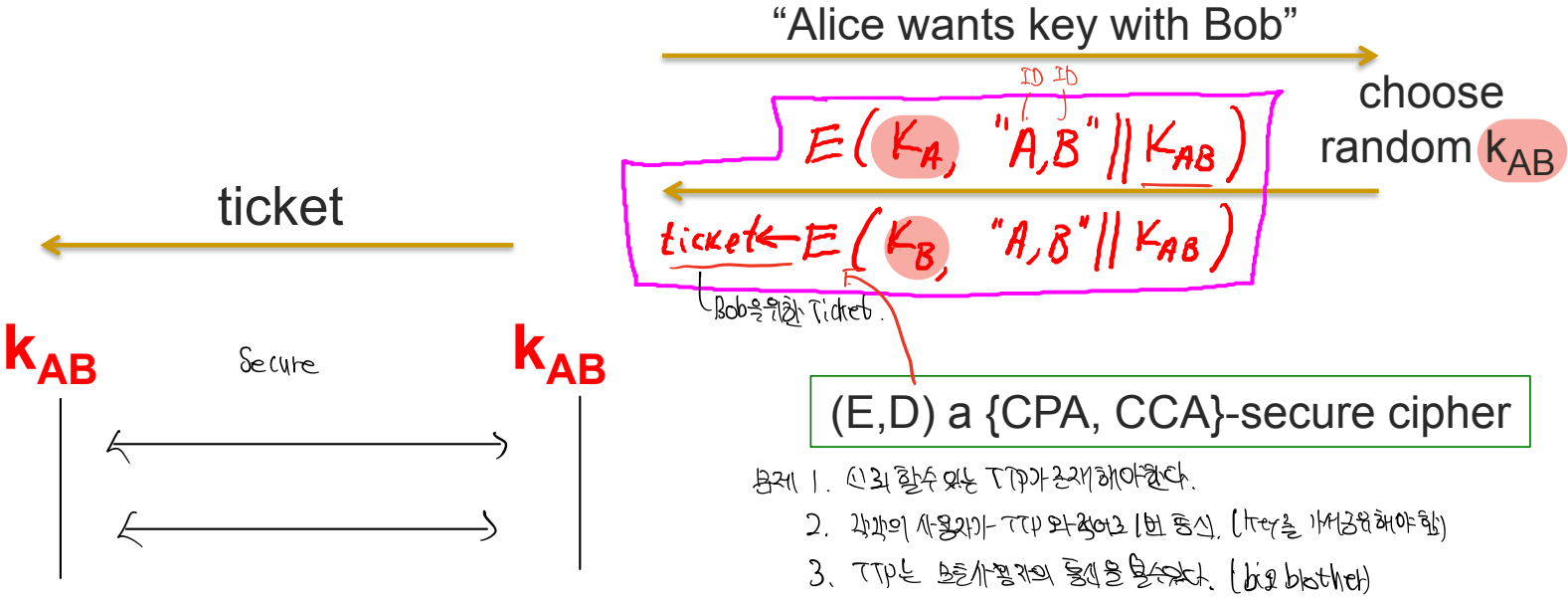
Every user only remembers one key.

장점 : 모든사용자는 자신의 key 만 [TTP와공유하려고] 관리하면된다. ⇒ 이외의사용자와 key공유X.

Generating keys: a toy protocol

Alice wants a shared key with Bob.

Bob (k_B) **Alice** (k_A) = TTP와 공유한 key **TTP**



Generating keys: a toy protocol

Alice wants a shared key with Bob. Eavesdropping security only.

Eavesdropper sees: $E(k_A,\ \text{“A, B”} \,||\, k_{AB})$; $E(k_B,\ \text{“A, B”} \,||\, k_{AB})$

(E,D) is CPA-secure $\Rightarrow$

eavesdropper learns nothing about k_{AB}

(E,D) is CCA-secure $\Rightarrow$ //

Note: TTP needed for every key exchange, knows all session keys.

(basis of Kerberos system, later)

Key question

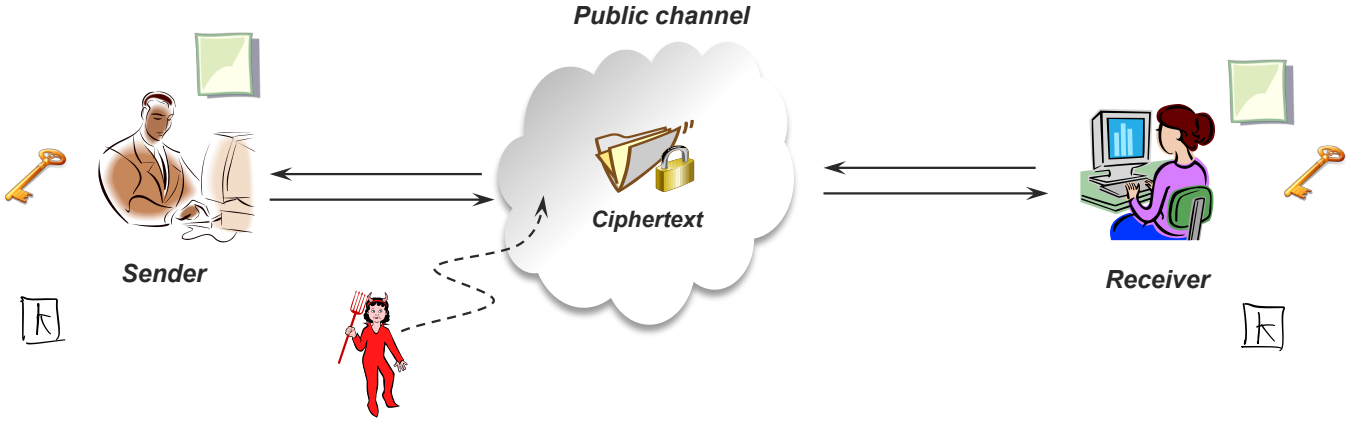
Can we generate shared keys without an **online** trusted 3rd party?

Answer: yes!

Starting point of public-key cryptography:

- Diffie-Hellman (1976), RSA (1977), ElGamel (1979)
- More recently: ID-based enc. (BF 2001), Functional enc. (BSW 2011)

Limitations of Symmetric-key Cryptography



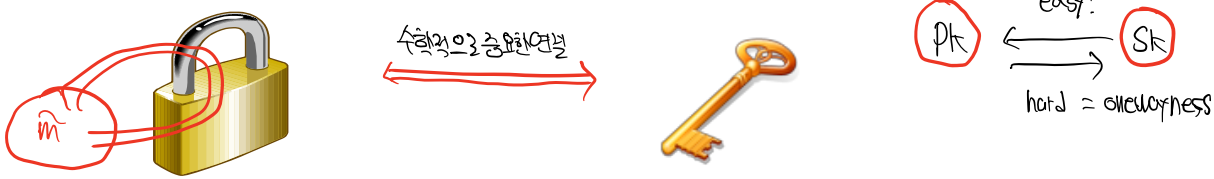
- Three problems that arise when using SKE (alone)
  - Key distribution
  - Managing so many secret keys, i.e., O(n) keys for each user
    - In 'closed systems' where physical means are possible
  - Inapplicability in 'open' systems

사용자는 자신안의 PK 와 SK만 가진다.
SK는 안전하게 저장하면 된다.
⇒ 해결

The Public-Key Revolution

- "New Directions in Cryptography" – Diffie and Hellman in 1976
  - Observed that there is a natural *asymmetry* in the world
  - Proposed three public-key primitives:
    - Public-key encryption
    - Digital signatures
    - Key exchange

- In public-key encryption schemes,
  - *Public key(PK) can be public (available even to adversary)*
  - How can PKE addresses the (three) limitations of SKE?



Public Key (PK)

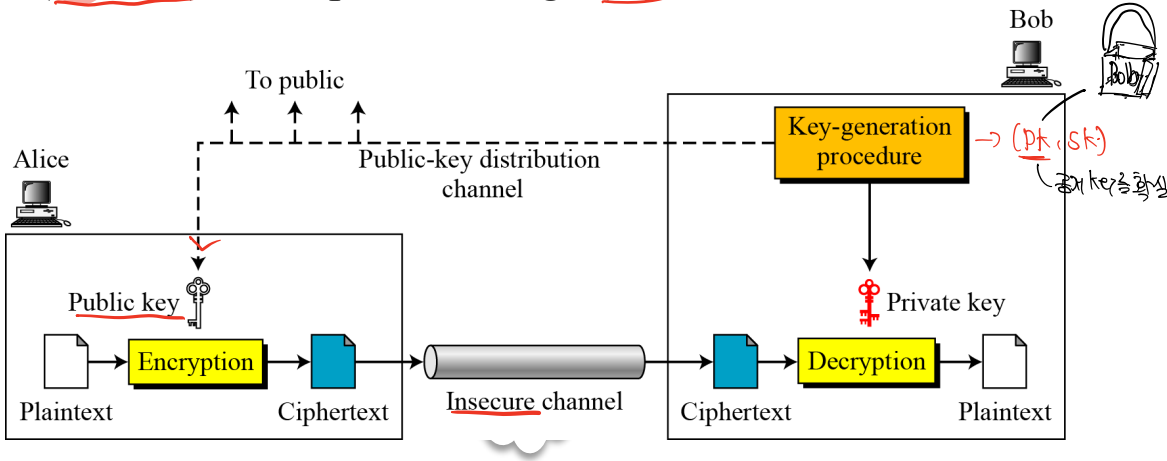
누구에게나 open (공개키 암호화)

Secret Key (SK)

열쇠를 가진 이만 자물쇠 open

Public-Key Encryption (PKE)

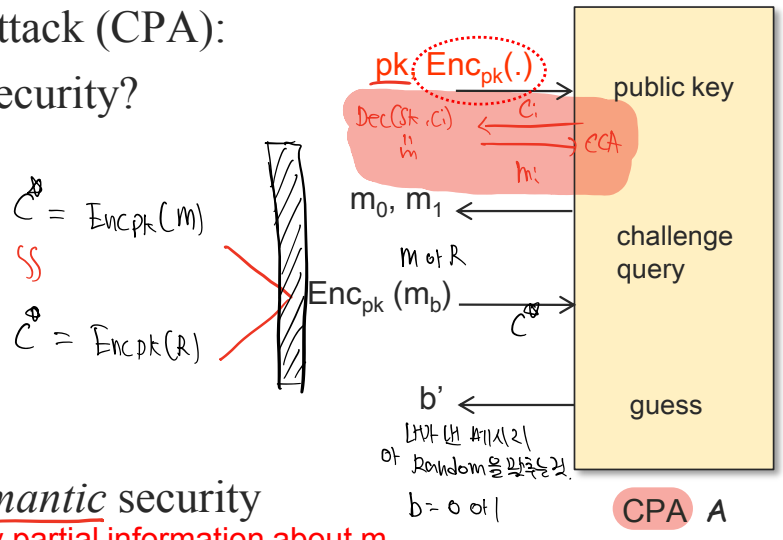
- Definition of PKE = (Gen, Enc, Dec)
  - Gen(1^n) → a public key (PK) and a private key (SK)
  - Enc(PK, m) → a ciphertext (CT)
  - Dec(SK, CT) → output a message (m)



  - PK is the receiver(Bob)'s public key
    - How can sender(Alice) obtain PK?
    - Need authenticated channel for distributing PK (why?)

Security of PKE

- Adversary is given a public-key PK, meaning that:
  - Adversary can encrypt messages of its choice (for free) (why?)
    - Can manipulate Enc algorithm at its will
  - Defining chosen-plaintext attack (CPA):
  - How about defining CCA-security?



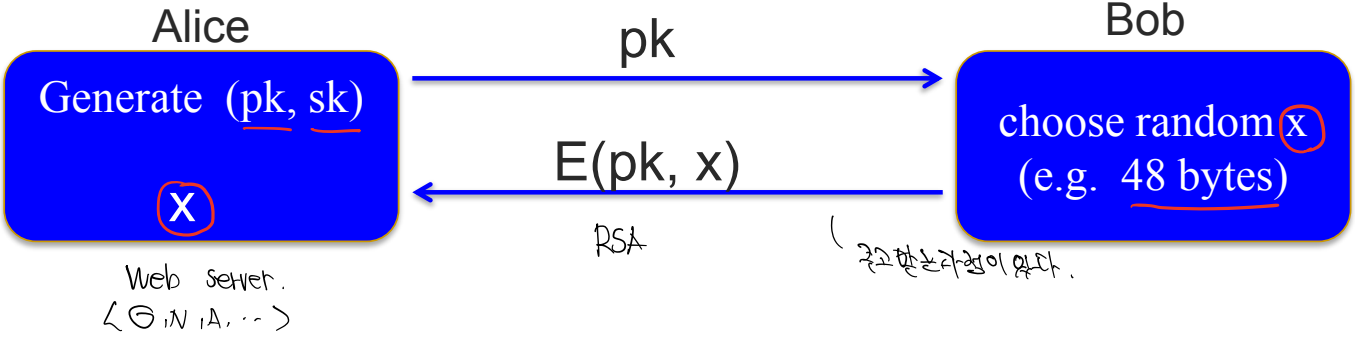
- We use the fact that:
  - IND-CPA security vs. *Semantic* security
  - Theorem:

Any partial information about m

> An encryption scheme is IND-CPA (or IND-CCA) secure iff an encryption scheme is semantically secure

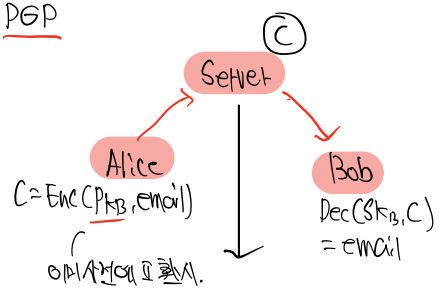
Applications

- **Key exchange** (e.g. in HTTPS)

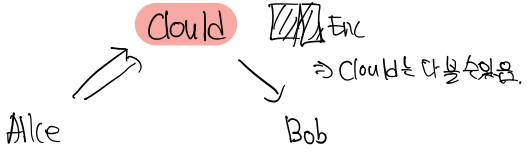


- **Non-interactive applications**: (e.g. Email)

  - Bob sends email to Alice encrypted using pk_{alice}
  - Note: Bob needs pk_{alice} (public key management)



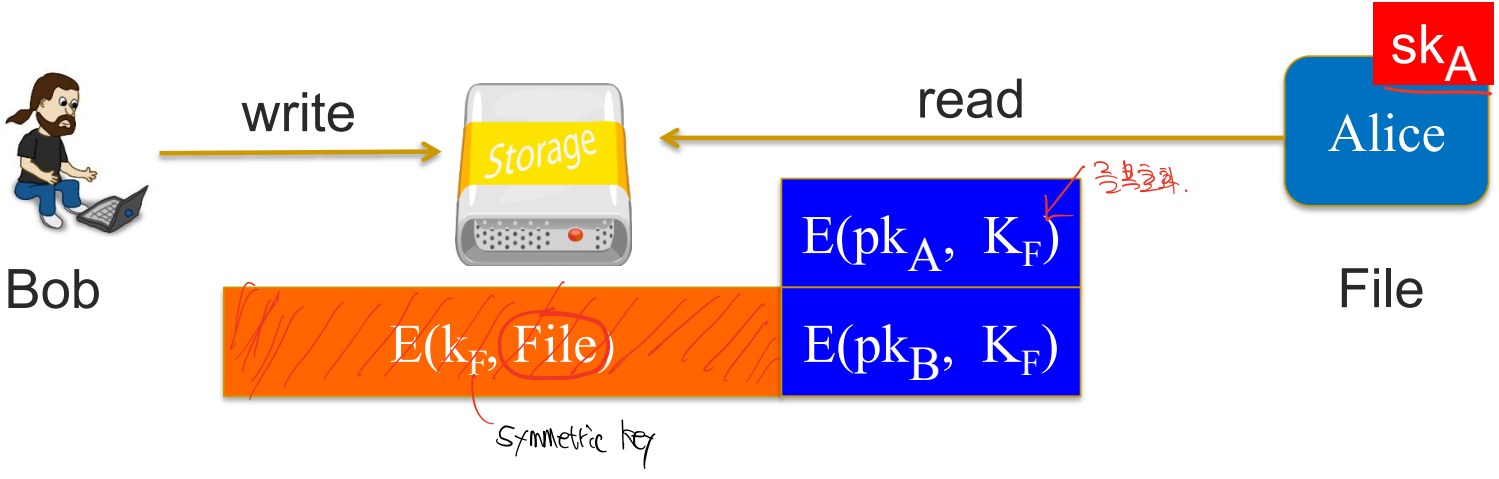
Applications



Key exchange (e.g. in HTTPS)

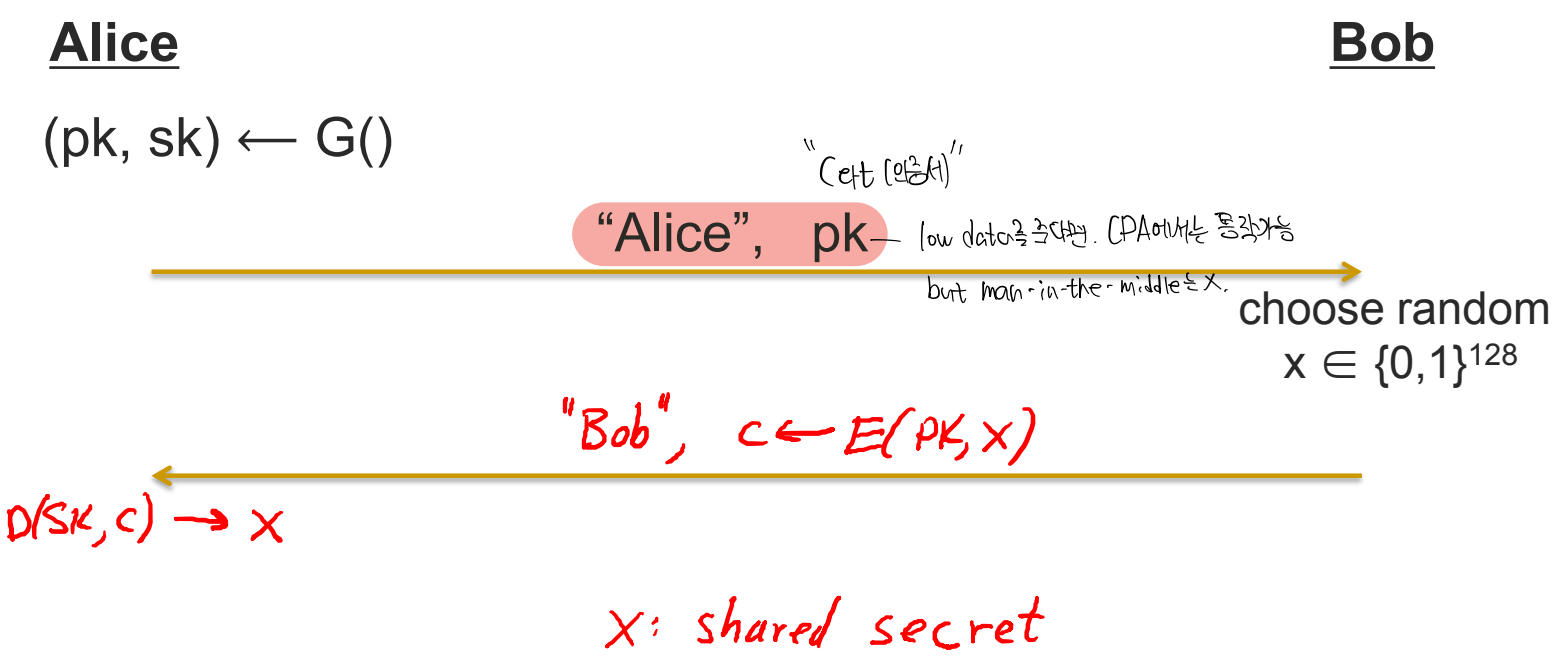
Encryption in non-interactive settings:

- Secure Email: Bob has Alice's pub-key and sends her an email
- Encrypted File Systems (e.g. secure Cloud)



Key Exchange(1)

- Establishing a shared secret



Key Exchange(2)

- Security (only eavesdropping)

Adversary sees **pk, E(pk, x)** and wants **x** $\in M$

Semantic security $\Rightarrow$

adversary cannot distinguish

{ pk, E(pk, x), x } from { pk, E(pk, R), R$\in$M }

$\Rightarrow$ can derive session key from E(pk, x).

Note: protocol is vulnerable to man-in-the-middle attack

Key Exchange(3)

- Insecure against man in the middle attack

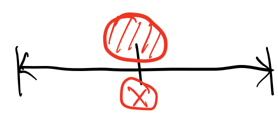
As described, the protocol is insecure against **active** attacks

Alice	**MiTM** (강력한 공격자)	**Bob**
(pk, sk) ← G()	(pk', sk') ← G()	
"Alice", pk →	≠	"Alice", pk' →
		Bob이 받는 곳 보니 Alice로 보냈다는 장치가 있어야 한다 = 인증서.
		choose random $x \in \{0,1\}^{128}$
✗ ← "Bob", E(pk, x)	✗	← "Bob", E(pk', x) (공격자의 PK.)

➔ How can we solve this problem?

통신은 가능하다 (방해 X)

두 사람의 통신 내용을 공격자가 감청할 수 있다.



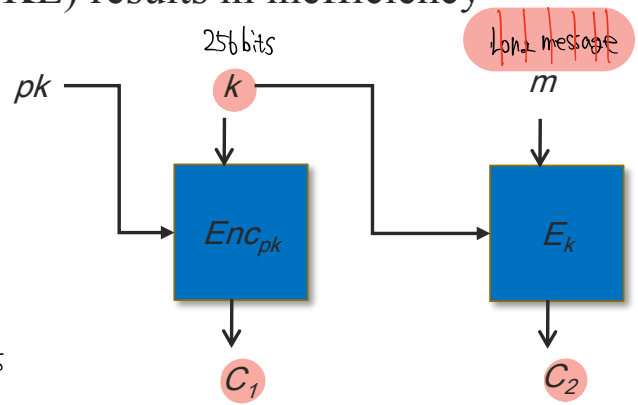
Encryption in Non-Interactive Settings

- SKE is significantly more efficient than PKE
  - Encrypting long messages (only using PKE) results in inefficiency

긴 메세지를 PK를 통해 암호화는 비효율적이다.

- Hybrid encryption (= PKE+SKE)
  - How to decrypt CT=(C_1, C_2)
  - Widely used in practice

$C = Enc(pk, m)$ 2048 bits 1000 bits



Construction of PKE^{hy} = (Gen^{hy}, Enc^{hy}, Dec^{hy})

1. $Gen^{hy}(1^n)$: runs $Gen(1^n)$ to output (pk, sk)
2. Enc^{hy}(pk, m) : on input (pk, m):
   (1) Choose a random k ← $\{0,1\}^n$
   (2) Compute $C_1 \leftarrow Enc_{pk}(k)$ and $C_2 \leftarrow Enc_k(m)$ and output $CT = (C_1, C_2)$

   SKE (대칭키)
3. Dec^{hy}(sk, CT) : on input (SK, $CT = (C_1, C_2)$):
   (1) Compute $k \leftarrow Dec_{sk}(C_1)$
   (2) Output the message $m \leftarrow Dec_k(C_2)$

RSA Encryption(1)

$n = p \cdot q$ ⇄ p, q ⇒ one-way! (hard ⇄ easy!)

현대의 p, q를 구할 수 있는 수학적으로 e→d 를 구할 수 있다.

⇒ 소인수 분해 등장. 암호화는 수학적으로 안전.

- "Textbook RSA" encryption (never use!) — 컴퓨터에서 해독 개념.
  - PK = (n, e) and SK = (d)
    - p, q = 1024 bits and thus n = 2048 bits (at the 112-bit security level)
  - Enc and Dec algorithms require only one exponentiation
  - *Deterministic* encryption → insecure !!
    - Same message incurs same ciphertext

PK $(n = p \cdot q, e)$ SK d

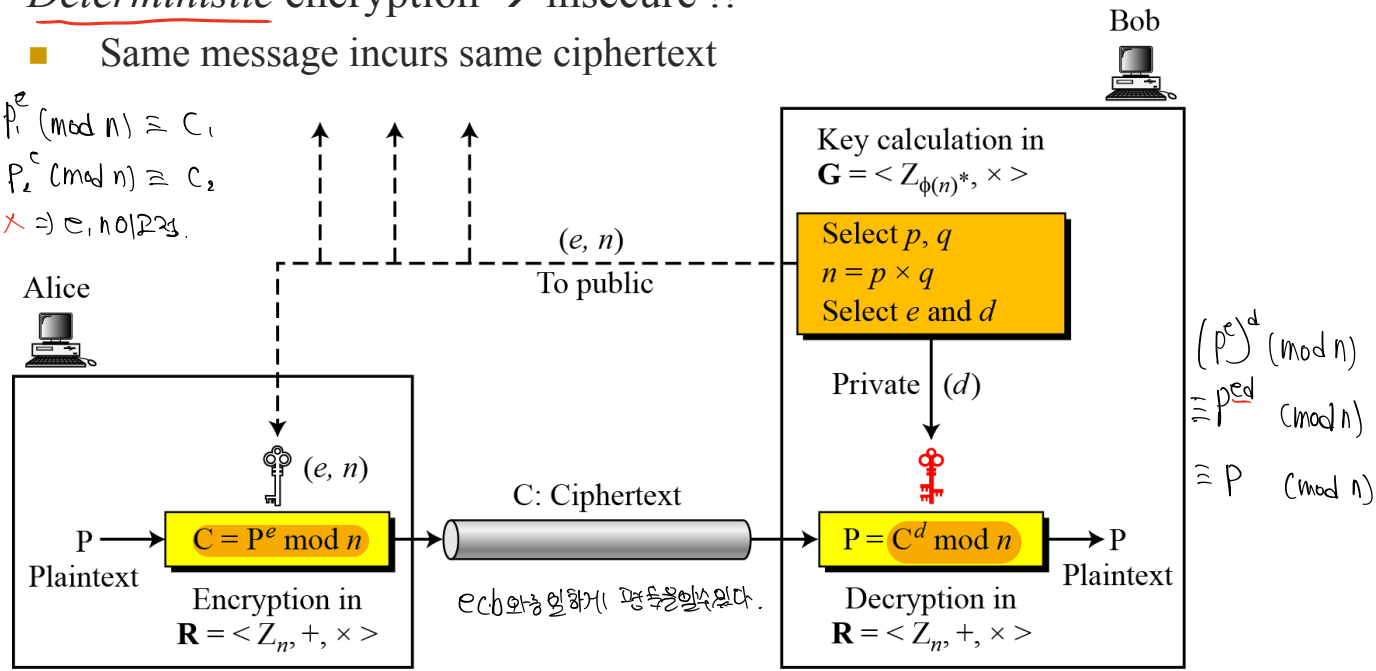
$e \cdot d \equiv 1 \pmod{\phi(n)}$: 지수에서 만나서 사라진다.

대칭키에서 112bit 사용과 같다.

P_1 = Hello! → $P_1^e \pmod n \equiv C_1$

P_2 = Monday → $P_2^e \pmod n \equiv C_2$

⇒ random X ⇒ e, n이 고정.



ECC와 같이 암호화 할 수 있다.

$(P^e)^d \pmod n$

$= P^{ed} \pmod n$

$= P \pmod n$

RSA Encryption(2)

- Key generation

RSA_Key_Generation

{

Select two large primes p and q such that $p \neq q$.

$n \leftarrow p \times q$

$\phi(n) \leftarrow (p-1) \times (q-1)$

Select e such that $1 < e < \phi(n)$ and e is coprime to $\phi(n)$

$d \leftarrow e^{-1} \bmod \phi(n)$ // d is inverse of e modulo $\phi(n)$

Public_key $\leftarrow$ (e, n) // To be announced publicly

Private_key $\leftarrow d$ // To be kept secret

return Public_key and Private_key

}

$(p^e)^d \equiv p \pmod{n}$

- Number theory behind RSA

If $n = p \times q$, $a < n$, and k is an integer, then $a^{k \times \phi(n)+1} \equiv a \pmod{n}$.

$P_1 = C^d \bmod n = (P^e \bmod n)^d \bmod n = P^{ed} \bmod n$

$ed = k\phi(n) + 1$ // d and e are inverses modulo $\phi(n)$

$P_1 = P^{ed} \bmod n \rightarrow P_1 = P^{k\phi(n)+1} \bmod n$

$P_1 = P^{k\phi(n)+1} \bmod n = P \bmod n$ // Euler's theorem (second version)

Modular Exponentiation: m^e, C^d (mod n)

- Very big numbers when computing m^e & C^d (mod n)
  - For instance, |m| & |C| = 2048 bits
  - |e| or |d| = 2048 bits

- How to perform modular exponentiation in an efficient way?
  - Square-and-multiply method
  - Idea: if we calculate a^{32} (mod n), then we can compute

$$a \longrightarrow a^2 \longrightarrow a^4 \longrightarrow a^8 \longrightarrow a^{16} \longrightarrow a^{32}$$

in 5 steps by repeated squaring

  - Can compute a^b (mod n) in k steps when $b=2^k$
  - But what if b is not a power of 2 ?

$a \cdot a = a^2$
$a^2 \cdot a^2 = a^4$
$a^4 \cdot a^4 = a^8$
$a^8 \cdot a^8 = a^{16}$
$a^{16} \cdot a^{16} = a^{32}$

5번의 곱셈.

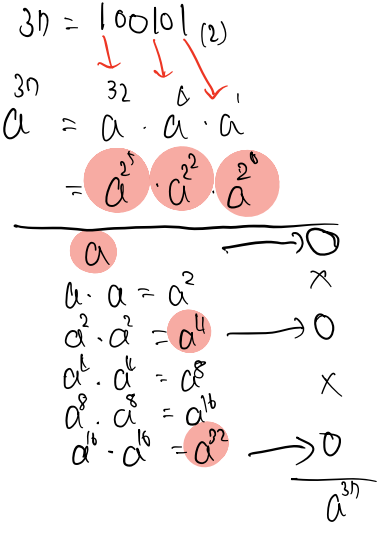
$\therefore a^{2^k}$ 이면 k번 제곱한다.

Square-and-multiply method

- Algorithm:

  Let $\text{bin}(n) = b_{k-1} \ldots b_0$ be the binary representation of n, meaning

$$n = \sum_{i=0}^{k-1} b_i 2^i$$

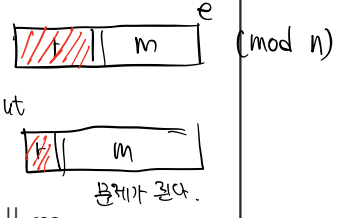


  Alg $\text{EXP}_G(a, n)$ // $a \in G$, $n \geq 1$
  $b_{k-1} \ldots b_0 \leftarrow \text{bin}(n)$
  $y \leftarrow 1$
  for $i = k - 1$ downto 0 do $y \leftarrow y^2 \cdot a^{b_i}$
  return y

- Can be applied to other schemes that require modular exponentiation

Padded RSA

- Textbook RSA is insecure under CPA (or CCA) adversary
- An improved approach: padded RSA
  - Gen(1n): output PK =(n, e) and SK=(d)
  - Enc(PK, m):
    - Choose a random string $r \leftarrow \{0,1\}^{|n|-l}$
    - Output a ciphertext $C = (r \| m)^e \pmod n$
  - Dec(SK, C):
    - Compute $r \| m = C^d \pmod n$
    - Output message m by taking the l LSBs of $r \| m$

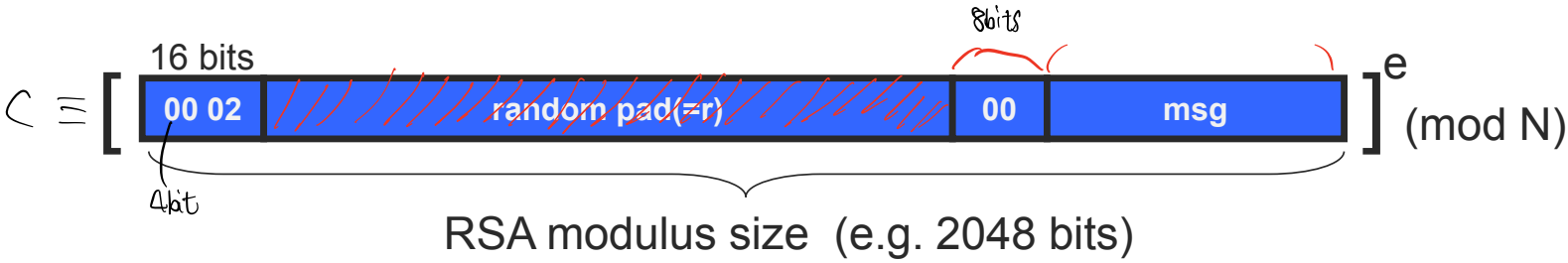


- Note:
  - If message length l is too large, padded RSA is not CPA-secure (why?)
  - If l is very small, it is proven that

    Padded RSA is CPA-secure under RSA problem

PKCS #1 v1.5

- PKCS(Public-Key Cryptography Standard)#1 version 1.5
  - A widely-used and standardized encryption (e.g. in HTTPS)
  - Utilize preprocessing, similar to (essentially) padded RSA encryption



  - r is a randomly-generated string (where |r| is at least 46 bytes)
  - PKCS #1 v1.5 is believed to be CPA-secure, although no proof
  - Subsequently, it is shown that PKCS #1 v1.5 is not CCA-secure
    - Motivate a change to a new standard called 'RSA-OAEP' PKCS#1 ver 2.
    - But, PKCS#1 v1.5 is still widely deployed for backwards-compatibility

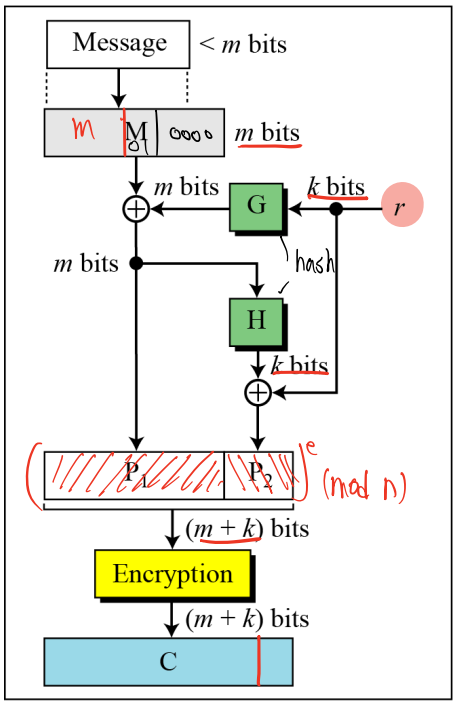
PKCS#1 v2.0: RSA-OAEP

- Optimal Asymmetric Encryption Padding (OAEP)
  - New preprocessing function

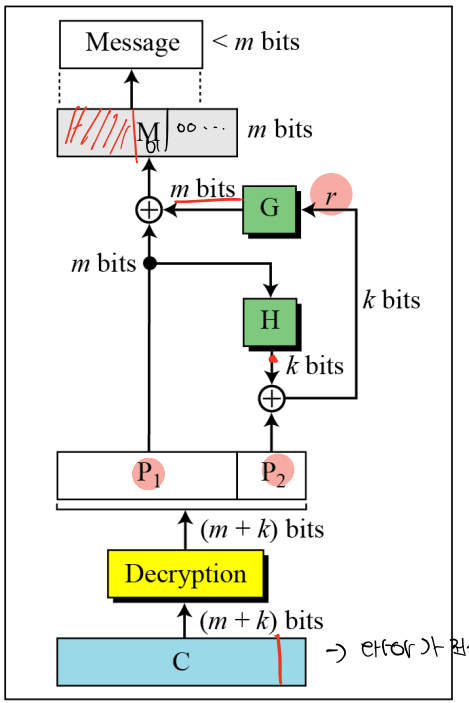
M: Padded message
r: One-time random number

P: Plaintext ($P_1 \| P_2$)
C: Ciphertext

G: Public function (k-bit to m-bit)
H: Public function (m-bit to k-bit)



Sender



Receiver

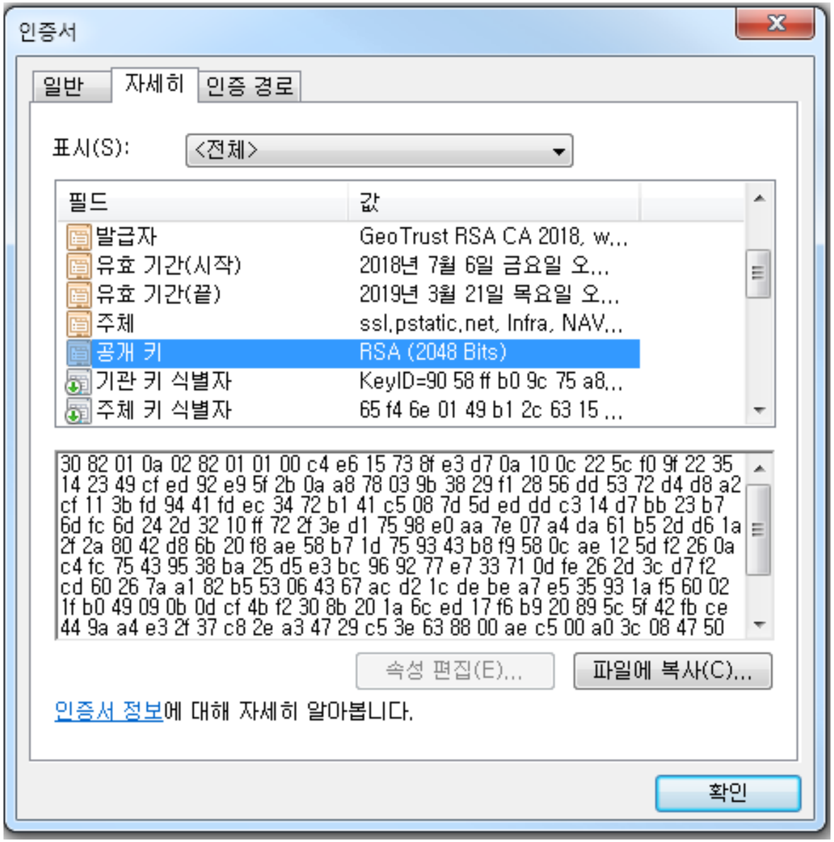
M = msg | 01 | 00..0

Check pad on decryption
→ reject CT if invalid

In practice, use SHA-256 for H and G

Under RSA assumption, RSA-OAEP is CCA-secure

Example of RSA Public-Key



01 00 01 = e

RSA With Low public exponent

$e \cdot d \equiv 1 \pmod{\phi(n)}$ (달라지면 d도 달라진다.)

$pk = (n, e)$ 2048bit

To speed up RSA encryption use a small e: $c = m^e \pmod N$

- Minimum value: **e=3** (gcd(e, φ(N)) = 1)
- Recommended value: **e=65537=$2^{16}+1$** 17bit 1 0000 0000 0000 0001 (2)

Encryption: 17 multiplications

$d \Rightarrow$ 2048 bits.

$m = c^d \pmod n$ 제곱승으로 선수조사하면된다.

성질. Asymmetry of RSA: fast enc. / slow dec.

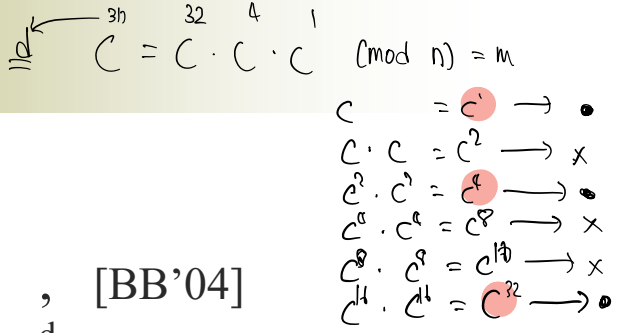
- ElGamal encryption (next): approx. same time for both.

NIST-recommended Key Lengths

Security of public key system should be comparable to security of symmetric cipher:

Cipher key-size		RSA Modulus size
80 bits	⟷	1024 bits
112 bits	⟷	2048 bits ← 에서 key 생성.
128 bits		3072 bits
256 bits (AES)	⟷	**15360** bits

Implementation attacks



Timing attack: [Kocher et al. 1997] , [BB'04]

The time it takes to compute $c^d \pmod N$ can expose d

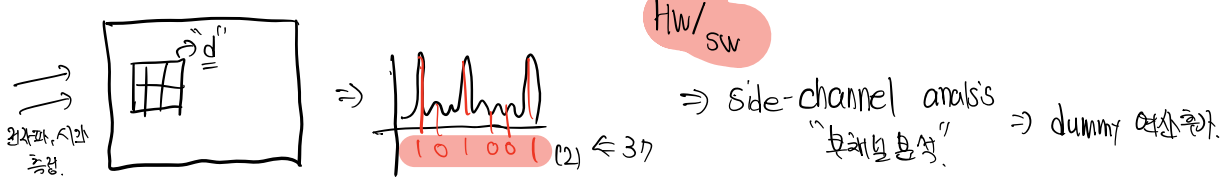
Power attack: [Kocher et al. 1999)

The power consumption of a smartcard while it is computing $c^d \pmod N$ can expose d.

Faults attack: [BDL'97]

A computer error during $c^d \pmod N$ can expose d.

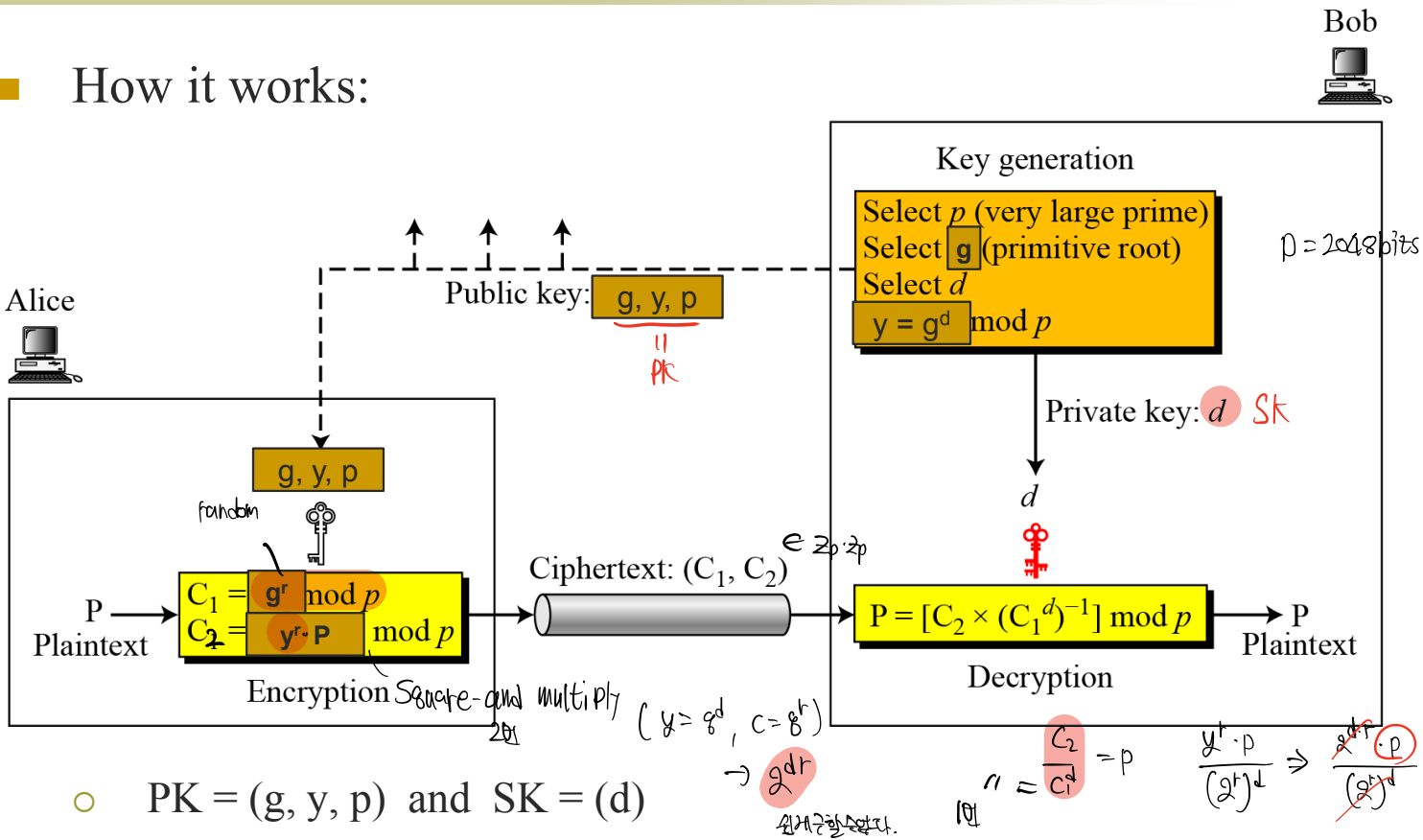
A common defense:: check output. 10% slowdown.



ElGamal Encryption (1)

$y = g^x \pmod p$
given (y, g, p) find x ⇒ 이산대수문제 (어려움)

- How it works:



- PK = (g, y, p) and SK = (d)
- CT is comprised of two group elements (C_1, C_2)
- Enc requires two Exps and Dec requires one Exp

ElGamal Encryption (2)

$c = (r \| m)^e \pmod n$ = padded

- Security of ElGamal encryption
  - Theorem:

$c_1 = g^r \pmod p$ $\quad y = g^x \pmod p$

$c_2 = y^r \cdot m \pmod p$

one time pad처럼 동작

> If Decisional Diffie-Hellman(DDH) assumption is hard in G, then ElGamal encryption is CPA-secure (수동적인 공격자)

DDH (give (g, g^x, g^y, T), decide it $T = g^{xy}$ or T = random)

- Implementation issue:
  - Sharing public parameters (G, p, g) is possible (in contrast to RSA)
  - Recommended that p be at least 2048 bits
  - Comparison to RSA in terms of |CT| and encryption cost
  - But, when implementing ElGamal on Elliptic Curve group, we have that:
    - |G| = 224 bits (for 112-bit security level with RSA 2048 bits)
    - Why are smaller groups preferable?

PK = (n, e) SK = d

share 할까. ⇒ 별로 좋지 않음

타원곡선

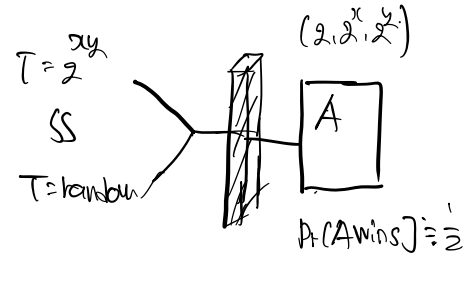
Group Size	Cost of Exponentiation
2^{160}	1
2^{1024}	260

224 bits

2048 bits

At least 5 times

DDH Problem



ElGamal system (DHIES/ECIES)

	RSA	ElGamal	(Ec)ElGamal	EcIES
Pk	(n, e) 2048 bits	(p, g, y) 2048 bits	(p, g, y) 224 bits "	
|CT|	$\|Z_n\|$ 2048	$Z_p \cdot Z_p$ 4096	$Z_p \cdot Z_p$ 448	
Enc	1 Exp	2 Exp	2 Exp	Ecrs
Dec	1 Exp	1 Exp	1 Exp	

Diffie-Hellman Integrated Encryption System

Elliptic Curve

- G: finite cyclic group of order n
- (E_s, D_s) : symmetric auth. encryption defined over (K,M,C)
- $H: G^2 \longrightarrow K$ a hash function

$pk = (g, p, h)$

$g^b = c_1$

ElGamal $\Rightarrow h^b \cdot p = c_2 / c_1^a$

Group.

E(pk=(g,h), m) :

$b \xleftarrow{R} Z_n$, $u \leftarrow g^b$, $v \leftarrow h^b \cdot p$

$k \leftarrow H(u,v)$, $c \leftarrow E_s(k, m)$

output (u, c)

(g^b, c)

D(sk=a, (u,c)) : $h = g^a$

$v \leftarrow u^a = (g^b)^a = g^{ab} = (h)^b$

$k \leftarrow H(u,v)$, $m \leftarrow D_s(k, c)$

output m

(EC) ElGamal Performance

pk(g, h, p)
(특정사람에게 사용자 소개)

E(pk=(g,h), m) :

$b \leftarrow Z_n$, $u \leftarrow g^b$, $v \leftarrow h^b$

D(sk=a, (u,c)) :

$v \leftarrow u^a$

Encryption: 2 exp. (fixed basis)

- Can pre-compute $\left[g^{(2^i)}, h^{(2^i)} \text{ for } i=1,\ldots,\log_2 n \right]$
- 3x speed-up (or more)

반복제곱식 테이블로 간직.

Decryption: 1 exp. (variable basis)

NIST-recommended Key Lengths

Suppose prime p is n bits long.

Best known algorithm (GNFS): run time $\exp(\tilde{O}(\sqrt[3]{n}))$

RSA n ElGamal p

cipher key size		modulus size	Elliptic Curve size
80 bits		1024 bits	160 bits
112 bits		2048 bits	224 bits
128 bits	⟷	3072 bits	256 bits
256 bits (AES)		**15360** bits	512 bits

EC

As a result: slow transition away from (mod p) to elliptic curves

Q & A